PlanWorks: A Debugging Environment for Constraint Based Planning Systems

Patrick Daley [*] [†] and **Jeremy Frank** and **Michael Iatauro** [‡] and **Conor McGann** [§] and **Will Taylor**
Computational Sciences Division
NASA Ames Research Center, MS 269-3
frank@email.arc.nasa.gov
Moffett Field, CA 94035

Introduction

Numerous planning and scheduling systems employ underlying constraint reasoning systems. Debugging such systems involves the search for errors in model rules, constraint reasoning algorithms, search heuristics and the problem instance (initial state and goals). In order to effectively find such problems, users must see why each state or action is in a plan by tracking causal chains back to part of the initial problem instance. They must be able to visualize complex relationships among many different entities and distinguish between those entities easily. For example, a variable can be in the scope of several constraints, as well as part of a state or activity in a plan; the activity can arise as a consequence of another activity and a model rule. Finally, they must be able to track each logical inference made during planning.

We have developed *PlanWorks*, a comprehensive system for debugging constraint-based planning and scheduling systems. PlanWorks assumes a strong transaction model of the entire planning process, including adding and removing parts of the constraint network, variable assignment, and constraint propagation. A planner logs all transactions to a relational database that is tailored to support queries for a variety of components. *Visualization* components consist of specialized views to display different forms of data (e.g. constraints, activities, resources, and causal links). Each view allows user customization in order to display only the most relevant information. Inter-view navigation features allow users to rapidly exchange views to examine the trace of the process from different perspectives. *Transaction query* mechanisms allow users access to the logged transactions to visualize activities across the entire planning process.

PlanWorks is implemented in Java and employs a MySQL relational database back-end. PlanWorks can be used either online while planning is performed, or offline after capturing the entire planning process. Furthermore, PlanWorks is an open system allowing for extensions to the transaction model to capture new planner algorithms, different classes of entity (e.g. complex resource classes) or novel heuristics. PlanWorks has been used to visualize logs generated by three different planners. PlanWorks was specifically developed for the Extensible Universal Remote Operations Planning Architecture (EUROPA$_2$) developed at NASA, but the underlying principles behind PlanWorks make it useful for many constraint-based planning systems.

The paper is organized as follows. We first describe some fundamentals of EUROPA$_2$. We then describe PlanWorks' principal components. We then discuss each component in detail, and then describe inter-component navigation features. We close with a discussion of how PlanWorks is used to find model flaws.

EUROPA$_2$

EUROPA$_2$ provides efficient, customizable *Plan Database Services* that enable the integration of automated planning into a wide variety of applications. These services are based on some simple building blocks. *Plans* are composed of *predicates*, each of which has a name, start time, end time, duration, and a (possibly empty) set of parameters. Each instance of a predicate in a plan is represented by a *token*, and each parameter of the predicate is represented by variables. Predicates are associated with *classes* that either represent *timelines* that support totally ordered sequences of states, or *resources* that support possibly concurrent actions that do not exceed a maximum capacity. Class instances are *objects*, and during planning each token is assigned to an object in the plan. *Domain rules* are assertions that if a predicate P is in a plan, then other predicates Q_i must also be in a plan, and are related to P by constraints among the variables of the predicates. Domain rules may also assert that resources are impacted by predicates; resource impacts are called *transactions*, and also have variables that represent them. EUROPA$_2$ does not implement any planning algorithm; rather, it provides services that support different planning algorithms according to the application.

A Sample Planning Domain

To illustrate the fundamentals of PlanWorks, we use a planning domain loosely based on a planetary surface robot named *Rover*. *Rover* is a mobile robot that can move from location to location. A *Rover* has a battery on board, and can replenish its energy levels using solar power. Locations are described as follows:

[*] Authors listed in alphabetical order.
[†] De Anza College
[‡] QSS
[§] QSS

```
class Location {
  int x; int y;
  Location(int _x, int _y){
    x = _x; y = _y;
  }
}
```

The properties of the Rover are described as follows:

```
class Rover {
  predicate At{Location l;}
  predicate Going{Location from;
                  Location to;}
  Resource battery;
  battery = new Battery(10, 3, 30);
}
```

A domain rule in EUROPA$_2$ describing rover movement is:

```
Rover::Going{
  neq(to, from); // to != from
  meets(object.At a0);
  eq(a0.l, to);
  met_by(object.At a1);
  eq(a1.l, from);
  subgoal(object.battery.transaction tx);
  calcConsumption(tx.quantity, from, to);
  // Consume at the beginning
  eq(tx.time, start);
}
```

Finally, a problem instance for the Rover is:

```
Rover spirit = new Rover();
Location rock = new Location(1, 1);
Location hill = new Location(2, 3);
Location lander = new Location(5, 8);
goal(Rover.At A);
eq(A.l, rock); eq(A.object, spirit);
leq(A.start, 0); leq(0, A.end);
goal(Rover.At B);
eq(B.l, lander); eq(B.object, spirit);
leq(B.start, 0); leq(0, B.end);
```

Getting PlanWorks the Goods

During planning, EUROPA$_2$ reads the domain rules to determine if any of them are applicable given the current state of the plan. If so, new tokens, resource transactions, variables, and constraints are created, and the domain rule application is recorded. As the planner makes decisions, tokens can be assigned to timelines, transactions can be assigned to resource instances, variables can be assigned, and constraints can be enforced, leading to reductions in the domains of variables. Each of these activities is logged, and each entity is assigned a unique key that allows for the tracking of entities and their relationships during planning. This information is passed down to PlanWorks to enable users to uncover the relationships between entities in the plan.

PlanWorks can be used in one of two modes. Planners can generate logs for PlanWorks offline, after which PlanWorks is invoked to view the logs. When planning takes a long time, this is impractical. Alternatively, PlanWorks can be started and provided a pointer to a planner. PlanWorks then allows the user to interleave planning and debugging. The user can run the planner for a fixed number of steps, investigate, then continue running the planner or terminate planning.

PlanWorks Components

Initial Views

Upon startup, PlanWorks presents users with a menu bar offering features for project creation and management. A project contains numerous *planning sequences* corresponding to the execution of a planner on a problem instance. The menu allows users to create new projects, add and delete sequences, and open a sequence for viewing.

When a sequence is opened for viewing, PlanWorks displays two views: the *Sequence Step View* and the *Sequence Query View*. The Sequence Step View, shown in Figure **??**, is a broad overview of the planning process. The view is presented as an inverted histogram, broken into components representing the number of variables, constraints and tokens in the plan. Moving the mouse over a bar of the histogram shows the step number and number of entities of each type in the plan. At a glance, the user sees how the plan's size evolved throughout planning, and can see patterns (such as thrashing in a chronological backtracking algorithm, or local optimal in a local search planner). An indicator above each bar of the histogram shows whether the data for that step has been loaded into PlanWorks.

The Sequence Query View allows the user to request detailed information about the underlying transactions over the entire planning sequence. The scope of logging is crucial to support these queries. Each time an entity is created or destroyed during planning, this information is logged; at creation time, each entity is given a unique key. These keys make it possible to track entities over the course of planning. Constraints can be tracked when they execute, tokens can be tracked as their state changes (e.g. from creation to insertion on an object), planner decisions can be tracked, and so on.

Examples of supported transaction queries include entity creation, assignments and unassignments of tokens to objects, assignments and unassignments of values to variables, constraint enforcement, checking for variables with only one domain value remaining, and more.

The Sequence Step View is also used to launch numerous other views of the plans generated at each step of the sequence. These views fall into one of three categories: *Plan Views, Entity Relationship Views* and *Transaction View*. These views are described further below.

Plan Views

Plan Views are holistic views of the entire plan. Plans are sequences of states or actions over time, so by their nature, the Plan Views are meant to convey a sense of what the plan looks like overall. However, EUROPA$_2$ can represent plans that are more complex than time-stamped sequences of actions. Plans can be *temporally flexible*; that is, states may have start times or durations that are unknown until plan execution. Further, plans may involve resources whose quan-

tities change over time. Thus, PlanWorks requires visual representations of timelines or resources that are temporally flexible. For this reason, three distinct Plan Views are provided.

The *Timeline View* is designed to show the sequence of predicates on a timeline. Since tokens can be unified, the Timeline View shows the number of unified tokens supporting each predicate; moving the mouse over the predicate shows the keys of the unified tokens and indicates which of these is the active token. The Timeline View shows the possible values of the start and end times of each predicate on the timeline. Finally, the Timeline View shows any free tokens. This is shown in Figure **??**.

The *Temporal Extent View* is designed to show more temporal information about tokens than the Timeline View. Each token has a series of icons representing the possible values of the start time (downward pointing triangles), end time (upward pointing triangles) and duration (horizontal line bracketed by the triangles). This is shown in Figure **??**. Moving the mouse over each of these shows the values, and an absolute time scale at the bottom is used for reference. By moving back and forth between the Timeline View and the Temporal Extent View, users can see how constraints on individual tokens lead to bounds and orderings on predicates in timelines. The Temporal Extent View also includes each resource transaction in the plan. Moving the mouse over the resource transaction shows the impact that transaction has on the resource. The *Resource Transaction View* restricts the Temporal Extent view so that only the resource transactions are shown.

The *Resource Profile View* shows the minimum and maximum quantities of a resource available as a function of time stemming from the transactions in the plan. Again, moving back and forth between the Temporal Extent View (or the Resource Transaction View) and the Resource Profile View, users can see how resource transactions lead to bounds on resource availability.

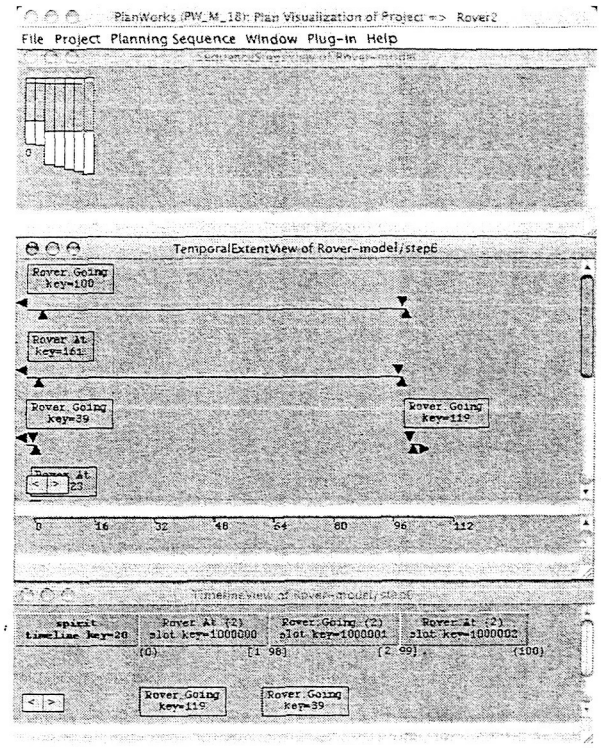

Figure 1: PlanWorks Plan Views. The Timeline View and Temporal Extent View are shown.

Entity Relationship Views

EUROPA$_2$ generates a large number of entities during the course of planning. These entities range from individual tokens, variables representing their parameters and constraints on those variables to object instances and domain rule invocation instances. The *Entity Relationship Views* are graphical views that show each of these entities and how they are related to each other. Under the Help menu, the Shapes option provides a handy guide to the shape each entity takes on in these views.

The *Navigator View* is an entity relationship graph capable of showing every entity in an individual plan. The Navigator View is launched by selecting an entity from any other view, and initially shows only a small number of entities and relationships. Each entity in the Navigator can be "opened" to show its relationships to other (currently hidden) entities, and subsequently "closed" to hide those relationships. Entities that can be closed are outlined in bold, and those that can be closed are not. The entity graph is directed; the description of the problem defines the initial set of entities, and all entities are derived from them via actions taken by the planner and the rules of the domain. The graph is acyclic, in that multiple relationships between entities apply. Users can explore the entities and their relationships and find out how various parts of a plan are related to each other. They may do so by hand, manually opening or closing various entities. The Navigator Window also supports a "Find Entity Path" feature that discovers paths between entities (whether they have been opened by the user or not). Navigator View also supports a "Find by Key" feature that hilights an entity with the given key. Each Rule Instance entity can be expanded to show the domain rule text that led to the new token or transaction, as well as the tokens involved in that part of the view.

The *Token Network View* restricts the Navigator view to only tokens, transactions and rule instances. This allows the user to focus exclusively on the "causal chain" that explains why particular tokens were generated. The resulting graph is a directed tree. As with the Navigator View, each Rule Instance entity can be expanded to show the domain rule text that led to the new token or transaction, as well as the tokens involved in that part of the view. This is shown in Figure **??**. The Token Network View also supports the same "Find Entity Path" and "Find by Key" features supported by the Navigator. Model constants may be the values of variables.

The *Constraint Network View* restricts the Navigator View to constraints, variables, tokens, transactions and model con-

stants. Initially, the view shows all model constants, predicates, transactions and rules. Model constants may be the values of variables, and may be complex structures; for example, in our simple Rover domain, *paths* consisting of an initial location, final location and cost in terms of energy consumption are constants. Each model constant can be opened to show its underlying structure. Tokens and transactions are associated with sets of variables, which in turn are in the scope of constraints. Domain rules may also have "local variables" to reduce the number of parameters of predicates. The user can incrementally explore the Constraint Network by opening tokens, transactions, or rules, and subsequently opening the variables or constraints. The Constraint Network View also supports the same "Find Entity Path" and "Find by Key" features supported by the Navigator.

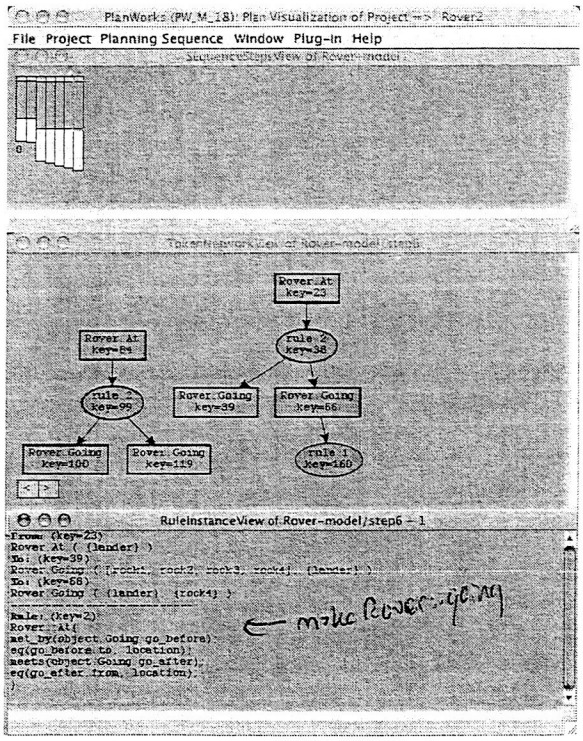

Figure 2: PlanWorks Token Network View and Rule Instance.

Transaction View

EUROPA$_2$ was designed to support multiple planning paradigms, from heuristically driven chronological backtracking planners to local searching planners to iterative sampling planners. Consequently, logging of information about how the planner makes decisions is the responsibility of the planner, while logging the consequences of planner decisions is the responsibility of EUROPA$_2$. The *Transaction View* shows every transaction EUROPA$_2$ performed this step. This includes checking domain rule applicability, entity creation and destruction, variable assignment, token state manipulation, constraint enforcement, and so on.

Navigating PlanWorks

An early decision was made in PlanWorks to create separate Views that contain information that users typically want grouped. However, PlanWorks contains numerous features that allow users to efficiently navigate between Views. These features allow users of PlanWorks to rapidly move from View to View when debugging planning domains and planners.

Launching Views

Almost all Views can be launched from any other View. The Navigator View can be launched when the mouse is over an entity such as a token, variable variable, constraint, constant or rule in a View (except the Transaction View). All other views can be launched when the mouse is over the background of a view.

Tracking Objects by Key

All objects have keys, and every view has a method to search for entities by key. Furthermore, the Views opened on starting PlanWorks have facilities for querying transactions by key. Finally, moving the mouse over entities will reveal the keys of entities. The entity relationship views can be used to provide keys used for querying the transaction database, for example.

Snapping to Entities

PlanWorks provides additional features to navigate between Views. An entity can be made *active* in one view, and the user can then "Snap to Active Entity" in a second view. This is especially useful for getting around the entity relationship views.

Filtering Views

In addition to manually opening and closing entity relationships to incrementally explore views, PlanWorks provides a custom filter for each View. This filter allows rapid reduction of the View to exclude designated predicates, classes (timelines or resources), or predicates in particular time ranges.

Overviews

Every View can have an associated Overview window that shows the view at a maximum zoom. This allows users to simultaneously examine a small number of related entities in a View, while also seeing the "Big Picture".

Stepping Forward and Backwards

All Views pertain to one step of the planning sequence. Each View has buttons that allow the user to advance or retreat the step the View shows. This permits a primitive "animation" feature that shows how a View changes during planning. As the View changes, the window is updated. Furthermore, the mouse allows users to either advance or retreat all Views simultaneously.

Managing Views

All Views are labeled at the top with View name, Project, Sequence number and Step. Thus, at a glance, a user can automatically tell what information they are seeing in a View. In addition, there is a drop-down menu named Window that allows users to see at a glance what windows are open, as well as either Tile or Cascade all open windows. Finally, using the mouse, users may automatically close, hide or open all Views.

Debugging in PlanWorks

We present two small examples of how PlanWorks can be used to find bugs in EUROPA$_2$.

Missing Model Rule

Suppose that the rule governing rover movement was missing a part:

```
Rover::Going{
  neq(to, from); // to != from
  meets(object.At a0);
  eq(a0.1, to);
  met-by missing
  ...
}
```

The resulting plan could then have two consecutive `Rover::Going` tokens; this would appear in the Timeline View. From here, the user could proceed in several ways. One option is to launch the Token Network View and see that only one `Rover::At` results as a subgoal from `Rover::Going`. Upon opening the Rule Instance View, the user would see the rule text and the context in which the rule was invoked, and be able to revise the rule. Alternatively, the user could launch the Navigator View and discover the problem.

The Wrong Constraint

As another example, supposer that the user used the wrong constraint in a rule:

```
Rover::Going{
  eq(to, from); // to == from? silly!
  ... }
```

In this case, the user might see an unexpectedly very long plan, which would appear in the Timeline View. Again, there are several candidate debugging scenarios. One option is for the user to open the Navigator View, and observe that `Rover::Going` begets `Rover::At` ad-infinitum. Opening a Rule Instance View on the `Rover::Going`, the user might notice the incorrect use of the `eq` constraint. Another possibility is that the user immediately suspects a problem with constraint reasoning, and opens the Constraint Network View. After finding the key of a parameter of a `Rover::Going`, the user then can check the transactions enforced on this variable using the Transaction View. Upon realizing that no `neq` constraints are enforced, the user can then check the Rule Instance View, and realize that the wrong constraint was used in the rule.

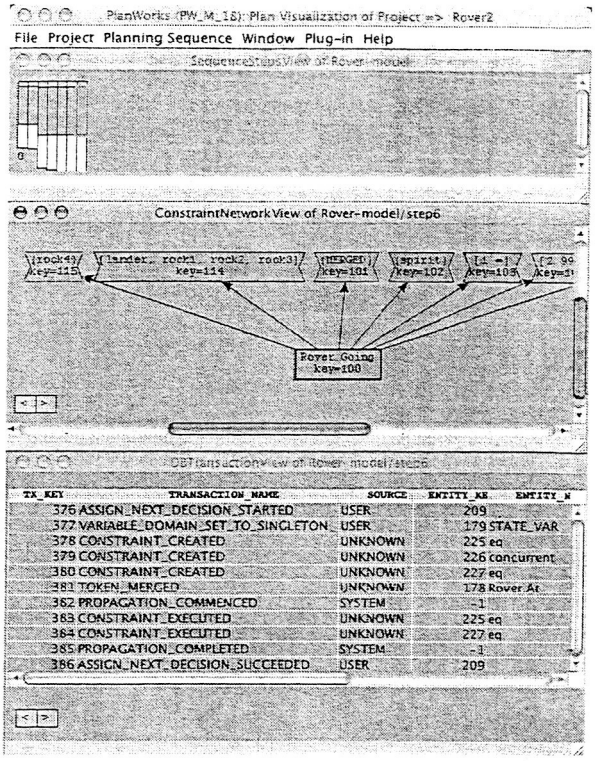

Figure 3: PlanWorks Constraint Network View and Transaction View.

Conclusions and Future Work

We have described PlanWorks, a system designed to debug planning domains and planners. While PlanWorks was designed to debug planners built on top of the EUROPA$_2$ system, it can be used by any planner that obeys a small number of rules about how to log its inner workings. PlanWorks was built largely with COTS technology, and has been fielded on Mac OSX 10.2 and 10.3, Linux and Solaris.

PlanWorks was originally conceived of as an Integrated Development Environment for building and managing projects with EUROPA$_2$. In the near future, PlanWorks will be extended to handle visual model building, visualizing plan execution and associated constraint reasoning. Furthermore, EUROPA$_2$ is designed to support many different planning algorithms. We will also extend PlanWorks to enable user customization to visualize different planner algorithms and heuristics.

Acknowledgements

The authors would like to acknowledge Andrew Bachmann's contributions to the NDDL language used to describe EUROPA$_2$ planning domains, Tania Bedrax Weiss for her ongoing work on EUROPA$_2$, Sailesh Ramakrishnan for his contributions as PlanWorks' prototype user, and Bob Kanefsky for the Potato prototype that ultimately evolved into PlanWorks. This project was funded by the NASA Intelligent Systems Program.